Certifiable Distributed Runtime Assurance of Distributed Real-Time Systems

Sagar Chaki and Dionisio de Niz

AIAA SciTech

January 10, 2017
Copyright 2017 Carnegie Mellon University

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Department of Defense.

NO WARRANTY. THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN “AS-IS” BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.

[Distribution Statement A] This material has been approved for public release and unlimited distribution. Please see Copyright notice for non-US Government use and distribution.

This material may be reproduced in its entirety, without modification, and freely distributed in written or electronic form without requesting formal permission. Permission is required for any other use. Requests for permission should be directed to the Software Engineering Institute at permission@sei.cmu.edu.

DM-0004363
Distributed Real-Time Systems of great relevance to aerospace community

- Single aircraft with multiple sub-systems
- Multiple UASs coordinating to achieve mission

Operate in uncertain and unknown environments

- External uncertainty – normal and denied environments
- Internal uncertainty – sophisticated components with unpredictable behavior, e.g., machine learning

Safety-critical & hard real-time requirements

- Failures can be catastrophic
- How do we verify & certify?
Runtime Assurance

Suppose we want to assure that system $S$ satisfies property $\Phi$

Key Idea:
• Add a runtime “enforcer” to observe the behavior of $S$
• Step in and take enforcement action to prevent violation of $\Phi$

Developed by AFRL & Barron Associates:
• https://www.cs.indiana.edu/~lepike/pubs/RTA-CPS.pdf
• Enforcer referred to as the “reversionary system”
Control theory – Simplex – CMU


Security Automata (Schneider) and Edit Automata (Ligatti et al.)


Prior Related Work (2)

Runtime Verification – specific safety properties


• Havelund, K. and Rosu, G., Monitoring Programs Using Rewriting, Proceedings of the 16th International Conference on Automated Software Engineering (ASE '01), November 2001

Limitations:

• single properties over single components
• enforcers implementations not formally verified
• restricted enforcer scheduling model
• enforcer runs at the same level as $S$
Our Project: Certifiable Distributed Runtime Assurance (CDRA)

Problem: Runtime assurance (RA) is critical for complex non-deterministic systems.

Key idea: monitor the system and take preemptive action to avoid unsafe states; monitors are simpler more verifiable.

Challenges:
- specifying safety policies rigorously;
- verifying monitor (aka enforcer) implementations;
- preventing unsafe inter-monitor interactions (single-node and distributed systems);
- protecting monitors from being circumvented.

Solution: A combination formal policy specifications, software verification, compositional reasoning, and verified hardware-supported isolation.
CDRA: Approach (1)

Controller Policy

Prove $E_1 \preceq P_1$ and $E_2 \preceq P_2$ using software verification

Controller Enforcer Implementation ($E_1$)

Controller Component ($C_1$)

Log: $\hat{\alpha}? \rightarrow \hat{\beta}!
\hat{\beta}? \rightarrow \alpha!
\alpha? \rightarrow \beta?
5s \rightarrow \hat{\gamma}!
< 5s \rightarrow \beta?

Logger Policy

Logger Enforcer Implementation ($E_2$)

Logger Component ($C_2$)

Log: $\hat{\gamma}? \rightarrow \hat{\delta}!
\hat{\delta}? \rightarrow \gamma!
\gamma? \rightarrow \hat{\gamma}?
< 2s \rightarrow \delta?
2s \rightarrow \hat{\delta}!

Node

${\{\alpha?, \beta!\}}$  ${\{\gamma?, \delta!\}}$
CDRA: Approach (2)

Verify $P_1 \parallel P_2 \leq P_N$ using assume-guarantee

Scale: (i) assumptions are simpler; (ii) abstract away unnecessary components; (iii) prove hierarchically.

Controller Assumption

$P_1 \leq A_1$  $P_2 \leq A_2$  $A_1 \parallel A_2 \leq P_N$

\[ P_1 \parallel P_2 \leq P_N \]
CDRA: Approach (3)

\[ P_1 \Rightarrow A_1 \quad P_2 \Rightarrow A_2 \quad A_1 \parallel A_2 \Rightarrow P_N \]

\[ P_1 \parallel P_2 \Rightarrow P_N \]

Verify \( P_1 \parallel P_2 \Rightarrow P_N \) using assume-guarantee

Scale: minimal system re-verification needed when a policy or enforcer is modified.

Controller Assumption

\[ \lessdot \]

\[ \lessdot \]

\[ \lessdot \]

\[ \lessdot \]
CDRA: Approach (4)

Verify $P_1 \parallel P_2 \leq P_N$ using assume-guarantee rules.

(i) Other (circular) rules exist; (ii) Challenges – (a) proving rule soundness; (b) finding right assumption.

Controller Assumption

$p_1 \leq a_1 \quad p_2 \leq a_2 \quad a_1 \parallel a_2 \leq p_N$

$p_1 \parallel p_2 \leq p_N$
CDRA: Approach (5)

1: Verify $E_i \leq P_i$
2: Verify $P_1 \parallel P_2 \leq P_{N1}$
3: Verify $P_3 \parallel P_4 \leq P_{N2}$
4: Verify $P_{N1} \parallel P_{N2} \leq P_s$

System-level Policy

Node$_1$

Node$_2$

Re-verification

Change
Problem: Ensure that $E_i$ is not circumvented. Inlining $E_i$ in $C_i$ will not work.

Solution: Use a hypervisor to execute $E_i$ in an isolated environment. Prove correctness of isolation by verifying the hypervisor.


Challenges: Performance, Correctness
Demonstrate verified runtime assurance on a realistic scenario. Give red team full control over applications (e.g., root access to OS) Initially in simulation, eventually on a hardware platform.
CDRA Testbed
Parrot Minidrones
Optitrack Localization
Optitrack Localization
Questions?